SociaLite: An Efficient Query Language for Social Network Analysis

Jiwon Seo    Stephen Guo    Jongsoo Park    Jaeho Shin
Monica S. Lam

Stanford Mobile & Social Workshop
Social Network Analysis

• Social network services
  - Twitter
  - Facebook
  - Google+
  - LiveJournal
  - Last.fm
  - Flipboard

• Social network analysis is very common and important
Problem

- Lack of efficient graph query language
- SQL
  - Weak support of recursion
  - Relational representation
- Parallel Frameworks
  - MapReduce, Pregel/Giraph
SociaLite

- Datalog-based query language
- High-level semantics
  - High-level description of graph analysis
  - Optimizations
    - Parallelization
    - Approximation
Outline

- Datalog
  - Overview
  - Performance issues
- Sequential SociaLite
- Parallel SociaLite
  - Data distribution
  - Prioritization
- Approximate computation
  - Early termination
  - Bloom-filter table
Overview of Datalog

- Declarative query language
- Supports recursion, good for graph algorithms

Path(t, d) :- Edge(1, t, d).

Path(t, d) :- Path(s, d1), Edge(s, t, d2), d=d1+d2.

MinPath(t, $Min(d)) :- Path(t, d).
SSSP in Datalog

Path(t, d) :- Edge(1, t, d). \hspace{1cm} (1)

Path(t, d) :- Path(s, d1), Edge(s, t, d2), d=d1+d2. \hspace{1cm} (2)

MinPath(t, $\text{Min}(d)) :- Path(t, d). \hspace{1cm} (3)

Path(s, d), Edge(1, t, d), ... represent tuples in the table
⇒ Edge(1, t, d) represents tuples (1, t, d) in Edge table

Aggregation: $\text{Min}(d)$
⇒ Applies to d’s for all (t, d) in Path after groupby t.
Datalog SSSP Performance

- >10x slower than optimized Java
- Why slow?
  - Unnecessary computations
  - Unnecessary memory space

Path(t, d) :- Edge(1, t, d).
Path(t, d) :- Path(s, d1), Edge(s, t, d2), d=d1+d2.
MinPath(t, $Min(d)) :- Path(t, d).

Java, adjacency list

Datalog, relational representation
Extensions in Sequential SociaLite

- Recursive aggregate functions
- Tail-nested tables
Recursive Aggregate Functions

- Aggregate function inside recursion
- A meet operator
  - A binary operator that is idempotent, commutative, associative.
  - Induces a partial order \( \subseteq \)
  - e.g. minimum, maximum

\( \Rightarrow \) Prioritization \( \Rightarrow \) SSSP into Dijkstra’s

* Details in ICDE ’13 SociaLite: Datalog Extensions for Efficient Social Network Analysis*
Tail-nested Tables

- Generalization of adjacency list
- Hint in table declaration
  - e.g. Edge(int src, (int target)).
- Reduces memory footprint

Edge(int src, (int target)).

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
</tr>
</tbody>
</table>

instead of
Performance of Sequential SociaLite

- As fast as optimized Java

*Details in ICDE ’13 SociaLite: Datalog Extensions for Efficient Social Network Analysis*
Parallel SociaLite

- SociaLite compiler/runtime for
  - A multi-core machine
  - Distributed machines
Distributed System Architecture

- Master
- Slave
- Slave
- Slave

- Checkpointing
- Resuming

Distributed File System
Inside a Slave Node

receive

Manager

Worker

Table registry

Runtime Resources

Shard lock container

Table shard map

Rule dependency map

send
Data Distribution

- Specified in a table declaration
  - Indicated by a location operator [ ]
  - e.g. Path[\text{int } t]\text{(double} d)\text{).}

- Supported sharding
  - Range-based sharding
    - e.g. Path[\text{int } t:0..100]\text{(double} d)\text{).}
  - Hash-based sharding
Automatic Distributed Computation

Foo[int x:0..10](int y).
Bar[int x:0..10](int z).
Baz[int z:0..10](int y).
Foo[x](y) :- Bar[x](z), Baz[z](y).

A [0..5]

B [6..10]
Batch Communication

- Data stored in a temporary table, and sent as a single message

Added rules:

```
Foo[int x:0..10](int y).
Bar[int x:0..10](int z).
Baz[int z:0..10](int y).
Foo[x](y) :- Bar[x](z), Baz[z](y).

Bar[z](x, z) :- Bar[x](z).
Foo[x](y) :- Bar[z](x, z), Baz[z](y).
```
Parallelism for Multi-core

- **Subsharding**
  - Each sharded table further subsharded $32n$ ways
  - Each subshard is protected by its own lock
  - Tasks operate on a subshard, and dynamically scheduled
Recursive Aggregate Functions

- Parallel prioritization
- Priority buckets holding tuples having priority within delta (approximately)
- Generalization of delta-stepping algorithm
- Optimization with light/heavy edges applied
Evaluation

- Benchmark algorithms
  - Shortest Path
  - PageRank
  - Mutual Neighbors
  - Connected Components
  - Triangles
  - Clustering Coefficients
Evaluation

- A single multi-core
- Distributed machines
Multi-core Evaluation

- **Input graph**
  - Friendster (120M nodes, 2.5G edges)

- **Machine configuration**
  - Dual socket Intel Xeon E5-2670
  - 256 GB memory
  - 16 cores (8 cores/socket)

- **Strong-scaling**
Multi-Core Evaluation

- Shortest paths
- PageRank
- Mutual neighbors
- Connected components
- Triangle
- Clustering coefficients

Graphs showing speedup and ideal speedup for different algorithms.
Distributed Machine Evaluation

- Input graph
  - Synthetic graph (RMAT, Graph 500)
  - 8M, 16M, 33M, 67M, 134M, 268M nodes, 16 avg edges/node

- Machine configuration
  - Up to 64 Amazon EC2 high-performance instances
  - Dual socket Intel Xeon X5570, 23GB memory
  - 8 cores (4 cores/socket)
  - 10 Gigabit Ethernet network

- Weak-scaling
  - Increased input size for increased computation power (machine #)
Distributed Machine Evaluation

- Shortest paths
- PageRank
- Mutual neighbors
- Connected components
- Triangle
- Clustering coefficients
SSSP in Giraph

```java
void compute(Iterator msgs) {
    double mindist = isSource()? 0:Double.MAX_VALUE;

    while (msgs.hasNext())
        mindist = Math.min(mindist, msgs.next());

    if (mindist < getVertexValue().get()) {
        setVertexValue(mindist);
        for (Edge edge: getOutEdge.values())
            sendMsg(edge.getDestVertexId(),
                    new DoubleWritable(mindist +
                                       edge.getEdgevalue()));
    }
}
voteToHalt();

Plus, for efficient implementation, you need to implement

• Data structure for edge
• Data structure for serialization (Writable)

new DoubleWritable(mindist +
                   edge.getEdgevalue());
```
SociaLite versus Giraph

- Shortest paths
- PageRank
- Mutual neighbors
- Connected components
- Triangle
- Clustering coefficients

Exec Time (Sec.)

Exec Time (Min.)
Approximate Computation

- Early termination for recursive queries
  - Prioritization improves accuracy
Approximate Shortest Path

Accuracy vs. Relative Execution Time without prioritization.
Shortest Path (w/ & w/o prioritization)

Without Prioritization

With Prioritization
Approximate Table

- Tables represented with a bloom filter
- Bloom Filter
  - Probabilistic set data structure
  - Elements represented as bits in a bitmap
  - Efficiently computes set membership
    - Can have false positives, but not false negatives
Approximate Table

- Bloom-Filter Based Table

  Foaf(n, f) :- Friend(n, f). \hspace{1cm} (1)
  Foaf(n, ff) :- Friend(n, f), Friend(f, ff). \hspace{1cm} (2)
  FoafSum(n, Sum(a)) :- Foaf(n, ff), Attr(ff, a). \hspace{1cm} (3)

(2\textsuperscript{nd} column of Foaf table is represented with a bloom filter)
Bloom-Filter Based Approx.

Foaf(n, f) :- Friend(n, f).
Foaf(n, ff) :- Friend(n, f), Friend(f, ff).
FoafSum(n, $Sum(a)) :- Foaf(n, ff), Attr(ff, a).

<table>
<thead>
<tr>
<th></th>
<th>Exact</th>
<th>Approx</th>
<th>Comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exec Time (Min)</td>
<td>28.9</td>
<td>19.4</td>
<td>32.8% faster</td>
</tr>
<tr>
<td>Memory Usage (GB)</td>
<td>26.0</td>
<td>3.0</td>
<td>11.5% usage</td>
</tr>
<tr>
<td>Accuracy (&lt;10% error)</td>
<td>100.0%</td>
<td>92.5%</td>
<td></td>
</tr>
</tbody>
</table>
Summary

- **SociaLite**
  - a Datalog-based query language
  - High-level semantics makes it easy to implement core graph algorithms
  - Parallelization
  - Approximation
Questions?